Design of a Floating Point Fast Multiplier with Mode Enabled

Umer Nisar Misgar, Wasim Ahmad Khan, and Najeeb-ud-din

Abstract— **In this paper we synthesize and optimize the delay of carry save based multiplier in 65nm technology using field programmable gate arrays (FPGA's) with mode enabled. We present a revolutionary design for floating-point decimal multiplication that utilizes decimal carry-save addition to reduce the critical path delays and power dissipation. A multiplier that stores a reduced number of multiplicand, multiples and uses decimal carry-save addition in the iterative portion of the design is presented. This work proposes a refreshing design for carry save based multiplier. The proposed logic has been further optimized to reduce the delay. Glitches are eliminated by proper selection of topology**

Index Terms—**Carry Save, FPGA, Mode Enabled, Multiplier.**

I. Introduction

Due to rapidly growing system-on-chip industry, not only the faster units but also smaller area and less power has become a major concern for designing very large scale integration (VLSI) circuits. Digital circuits make use of digital arithmetic's. Among various arithmetic operations, multiplication is one of the fundamental operation used and is being performed by an adder. There are many ways to build a multiplier each providing trade-off between delays and other characteristics, such as area and energy dissipation. However no design is considered as superior. Carry-Save based Multiplier is one of the promising techniques in terms of speed. It provides a compromise between ripple carry adder and carry look-ahead adder, but to a lesser extent at the cost of its area.

Thee are certain applications, like 3D graphics and signal processing where performance of the system strongly depends on the speed of multiplications. This is due to the fact that, these applications need to support high multiplication intensive operations. Therefore, there has been much work on advanced multiplication algorithms and designs [1, 2, 3, 4, 5, 6, 7, and 8].

The performance of the multiplier depends upon, how addition is carried out. The type of addition chosen always determines tradeoff between various parameters e.g., speed and area, power and energy dissipation, complexity and chip density, etc. So, a single multiplier cannot be optimized for all these parameters. Therefore the design of the multiplier depends upon the application of the multiplier. In this work, we have concentrated on the pace factor and power dissipation in multiplication and we present an insight of a fast multiplier based on carry save technique with mode enabled.

Manuscript received December 2, 2008.

Umer Nisar Misgar was with ECE Dept, with National Institutional Institute of Technology, Srinagar India and is presently with Department of Process Automation – Composite Plants, ABB Ltd., Bangalore, Karnataka, India, 560 010 (corresponding author: +919901979884; fax:+911942474576 e-mail: umer.nisar@ gmail.com).

Wasim Ahmad Khan was with ECE Dept, National Institutional Institute of Technology, Srinagar India and is presently with Tata Tele Services, New Delhi, India (e-mail: wasim_nit@yahoo.co.in).

Najeeb-ud-din is with Electronics & Communication Engineering, National Institutional Institute of Technology, Srinagar India (email: hnds@rediffmail.com).

II. Method

The multiplication requires the addition of several summands. The addition acceleration process depends upon the reduction in the number of summands. Acceleration of the formation of summands and acceleration of the addition of these summands will also determine the speed. In this respect one of the technique, which speeds up the addition process is Carry-Save Addition (CSA) technique. This technique comprises of a string of full adder circuits of the normal sort, where the carry inputs are used for the third input number, and the carry outputs for the second output number. In multiplication, one buffer-adder or pseudo-adder is usually used, and storage is provided for two numbers. On each pass through the adder, the stored numbers and one multiple of the multiplicand are added and the resulting two numbers returned to storage. Thus, the time required varies linearly with the number of summands. In any scheme employing buffer-adders, the number of adder passes occurring in a multiplication before the product is reduced to the sum of two numbers, will be two less than the number of summands, since each pass through an adder converts three numbers to two, reducing the count of numbers by one.

A more significant reduction in delay or to improve the speed of the multiplication, we must group the summands in threes and perform carry-save addition on each of these groups in parallel to generate a set of Sum (S) and Carry (C) vectors in one full-adder delay. Next, we group all of the S and C vectors into threes, and perform carry-save addition on them, generating a further set of S and C vectors in one more full-adder delay. We continue with this process until there are only two vectors remaining. They can then be added in a ripple-carry or a carry-look-ahead adder [9] to produce the desired product.

Delay through the carry-save array is reasonably less than delay through the ripple-carry array as sum 'S' and carry 'C' vector outputs from each row are produced in parallel in one full-adder delay [10]. Figure-1 shows a set of 12 buffer-adders connected to take 14 summands (Z_0 to Z_{26}) and express their sum as the sum of two numbers in a purely combinational adder fashion, which has a considerable speed advantage.

In general, it has been calculated that approximately $1.7\log_2 k - 1.7$ levels of CSA steps [10] are needed to reduce k summands to 2 vectors, which, when added, produce the desired sum. Thus, it is clear that as the number of bits for which product is to be determined increases, the delay increases less proportionally in case of carry save multiplier as compared to other multipliers if proper device and topology is used.

III. Architecture Procedure

The block level representation of Floating Point Carry-Save multiplication is implemented, as shown in Figure 2. The n-Input pins supply normalized multiplicand (having implicit 1 to the left of binary point) and n-more supply multiplier bits. Instead of carrying out "AND" operation in each block itself, the ANDed bits are produced all at once in an ANDer block and then introduced at the proper position. These ANDed signals are then added in Carry-Save floating-point representation format as shown earlier in Figure 1, to get the final product. The ANDed signals are introduced at their proper position within the floating-point multiplier. This calculates unnormalized (i.e. without implicit 1 to the left of the binary decimal point) floating-point product. The product is then passed through the normalizer block. This block normalizes the product if required depending upon the control signal in the standard IEEE format as shown in Figure 3 [11]. The normalized product is then outputted through the mode selector, which depending upon the mode selected gives either n-bit precise output in 1 cycle or n-bit product extended over two clock cycles. In case the product needs to be normalized, the exponent out from the exponent block is accordingly adjusted. The device also raises flag bits indicating the state of the output.

In the proposed architecture, RESET clears the output in an asynchronous manner, ENABLE allows chip select and is active low. When ENABLE is kept high the device offers high input impedance at the input and thus does not load the data bus of the device to which this multiplier is connected. This feature allows the multiplier to be connected to the microprocessor and act as math co-processor where the address issued by microprocessor may be decoded to get the proper ENABLE signal. Multiplier has the capability of either giving 16 bit precise floating point output in a clock cycle.

Alternately, if the multiplier is connected to n/2 bit wide bus only, the time division multiplexing is carried out and you may get the n-bit output in two clock cycles. This feature can be activated by applying the suitable MODE signal. This multiplier has flag pins, which indicate the state of the output. The output is valid only if VALID is high. SIGN flag indicates the sign of the output. ZERO and OVERFLOW goes active high in the conditions of zero and overflow respectively.

IV. Simulation Results

In order to measure the performance of the proposed algorithm and its implementation, we have designed our algorithm using VHDL [12, 13] and synthesized it using Altera's Quartus II Software. Figure 4 (a), 4 (b) shows the Timing analysis of the proposed Carry Save Floating Point Fast Multiplier with Mode Select feature in Mode 0 and Mode-1 respectively compare to a simple Carry Save Multiplier having no mode selects option as shown in Figure 5.

The functionality of this multiplier has been fitted on the most architecturally advanced, high performance, low power FPGA's in the market place, which provides the ability to turn on the performance and turn down the power consumption wherever needed. Selectable Core Voltage and the latest in silicon process optimizations are also employed to deliver the industry's lowest power, high performance FPGA's.

The overall analysis of this multiplier is summarized (**Table 1**) after optimizing the design using proper topology and device settings.

Table 1. Comparison of various parameters

S. No.	Parameter	Simple Carry Save Based Multiplier	Proposed Carry-Save Based Multiplier with Mode Select
1.	Total Delay Encountered	28.771 ns	9.243 ns
2.	Permissible Clock Frequency	35.86 MHz	108.18 MHz
3.	Max. no. of Calculation in 1 Sec.	35,860,000	108,189,000
4.	Total Thermal Power Dissipation	391.32mW	373.17 mW
5.	Core Dynamic Thermal Power Dissipation	0.0mW	14.18 mW
6.	Core Static Thermal Power Dissipation	318.17mW	301.12 mW
7..	I/0 Thermal Power Dissipation	72.11mW	57.87 mW
8.	Total Logic Elements	102/12480	197/12480
9.	Total Pins	32/343	58/343

V. Conclusion

In this paper, we proposed a novel design for a Floating Point Fast Multiplier with Mode select option which could be used in designing the latest state of the art mathematical processors. Since present day technology uses high clock frequencies and low voltage power supplies, our design is a potential candidate for the same. The implementation of the algorithm with an architecture and logic design is presented wherein the Speed and complexity of the design are compared to other designs.

Also, the proposed design is an area efficient multiplier useful in decreasing silicon area used from the chip and consequently reduces the cost of the chip. The proposed design provides an simple alternative to the Booth Multiplier with speed that is at least similar to that of a Dadda Multiplier or Recursive Fast Multiplier, as the delay encountered is reduced to one-third (⅓) and the processing speed is increased to almost four (4) times than those obtained in other conventional techniques.

REFERENCES

[1] L. Dadda, "Some Schemes for Parallel Multiplier," Alta Frequenza, vol. 34, pp. 349-356, 1965.
[2] A. D. Booth, "A Signed Binary Multiplication Technique, Quarterly Journal of Mechanical and Applied Math., vol. 4, pp. 236-240, 1951.
[3] A. Farooqui and V. Oklobdzija, "General Data-Path Organization of A MAC Unit for VLSI Implementation of DSP Processors" Proceedings IEEE Int'l Symposium on Circuits and Systems, vol. 2, pages 260-263, 1998.
[4] P. F. Stelling, C. U. Mattel, V. G. Oklobdzija, and R. Ravi, "Optimal Circuits for Parallel Multipliers," IEEE Transactions on Computers, vol. 47(3): 273 - 285, 1998.
[5] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Computers, vol. 13 (2), pp.14-17, 1964.
[6] A. Weinberger, "4:2 Carry-Save Adder Module," IBM Technical Disclosure Bull, vol. 23, 1981.
[7] W. C. Yeh, and C. W. Jen, "High-Speed Booth Encoded Parallel Multiplier Design," IEEE Transactions on Computers, vol. 49 (7), pp. 692-701, 2000.
[8] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Transactions on Computers, vol. 45 (3), pp. 294-306, 1996.
[9] William Stallings, *Computer Organization and Architecture: Designing for Performance*, New Delhi, Prentice Hall, Sixth Ed. 2005.
[10] Carl Hamacher, Zvonko Vranesic, and Safwat Zaky, *Computer Organization,* McGrawHill, International Edition, 2002.
[11] Institute of Electrical and Electronics Engineers, "IEEE Standard for Binary Floating – Point Architecture," ANSI/IEEE Standard 754 –1985, August 1985.
[12] Kevin Skahill, *VHDL for Programmable Logic*, New Delhi, Pearson Education2nd Edition, 2006.
[13] Peter J. Ashenben, *The Designers Guide to VHDL,* CA, USA, Elsevier, Second Edition, 2005.

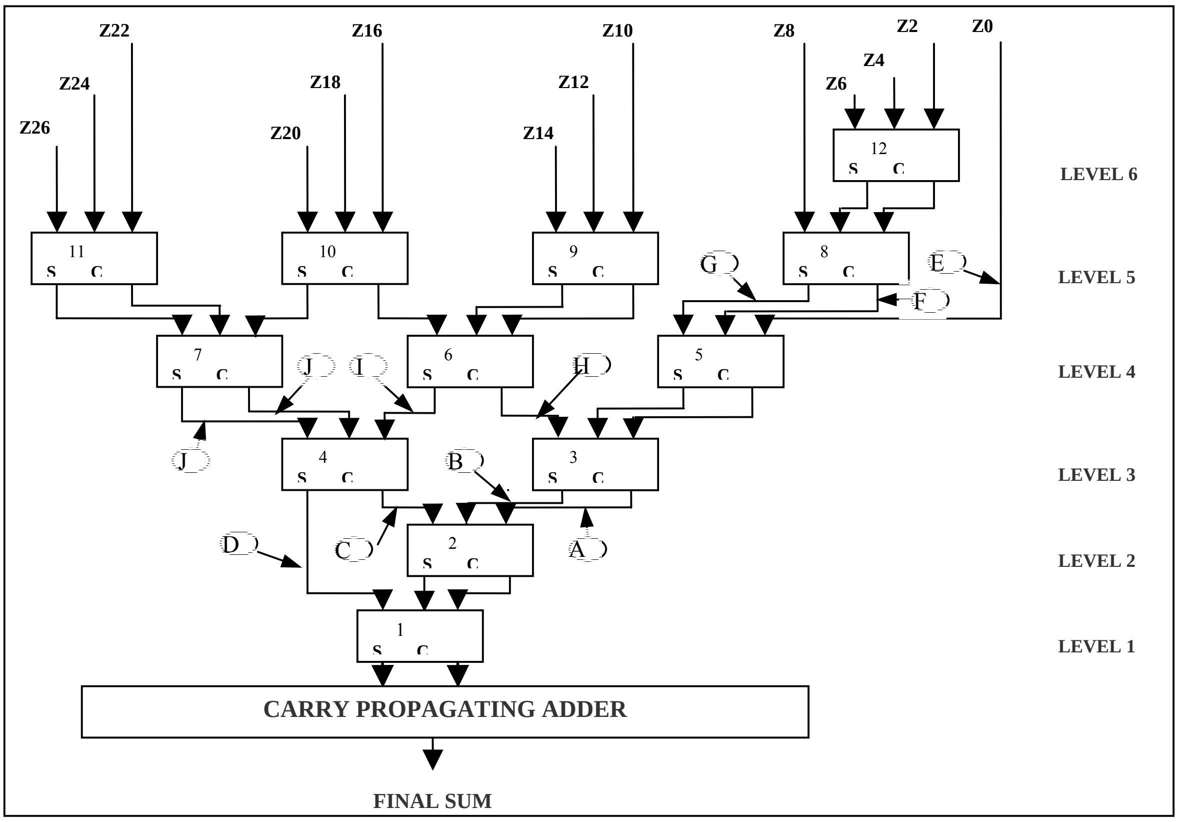


Figure 1. The adder tree

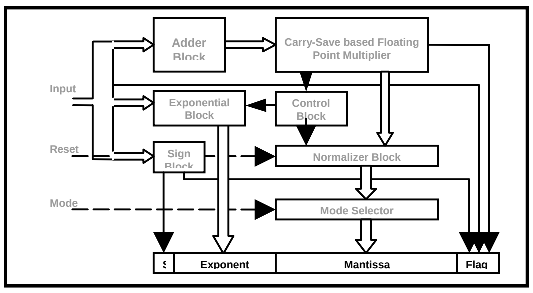


Figure 2. Proposed Mode Enabled Normalized Floating Point Multiplier

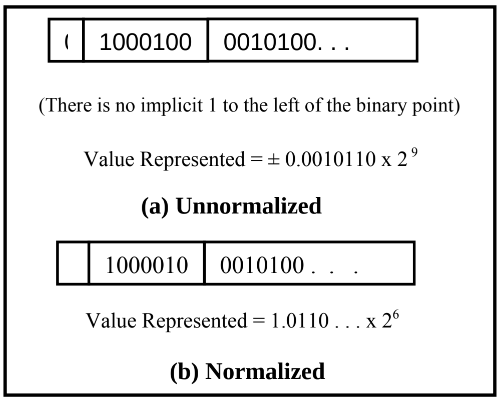
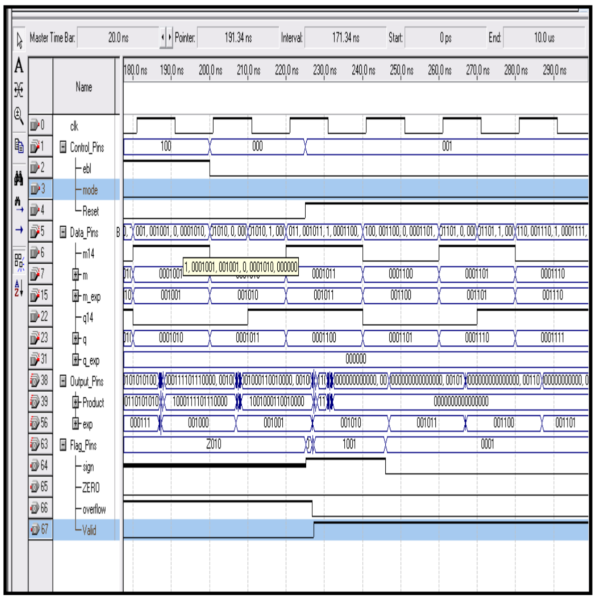

(1000100 0010100. . .
(There is no implicit 1 to the left of the binary point)
Value Represented = ± 0.0010110 x 2 9
(a) Unnormalized
1000010 0010100 . . .
Value Represented = 1.0110 . . . x 26
(b) Normalized

Figure 3. Floating Point Normalization in IEEE single precision format



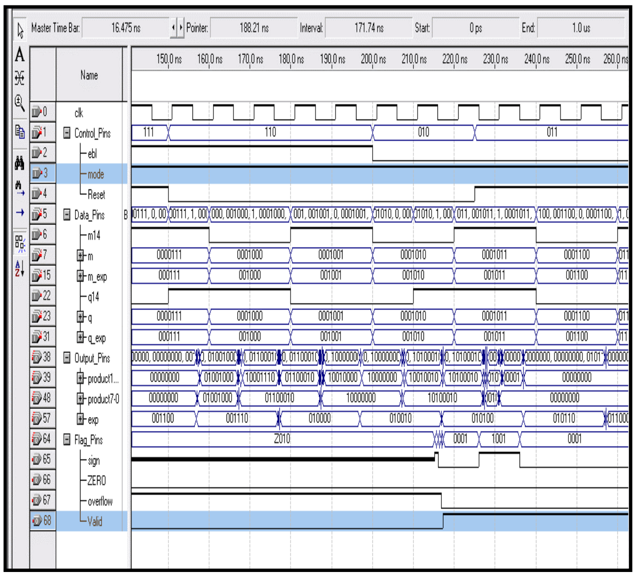


Figure 4(a). Desired product in one cycle, when 16-bit data bus is available – Mode 0.

Figure 4(b). Desired product in two cycles when 8-bit data bus is available - Mode 1.

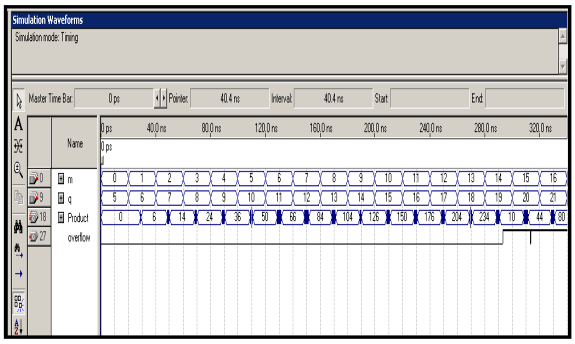


Figure 5. Showing the same desired product in a simple conventional technique – Mode 0 only.